# Comp 590-184: Hardware Security and Side-Channels

## Lecture 22: Hardware Bugs and Fuzzing

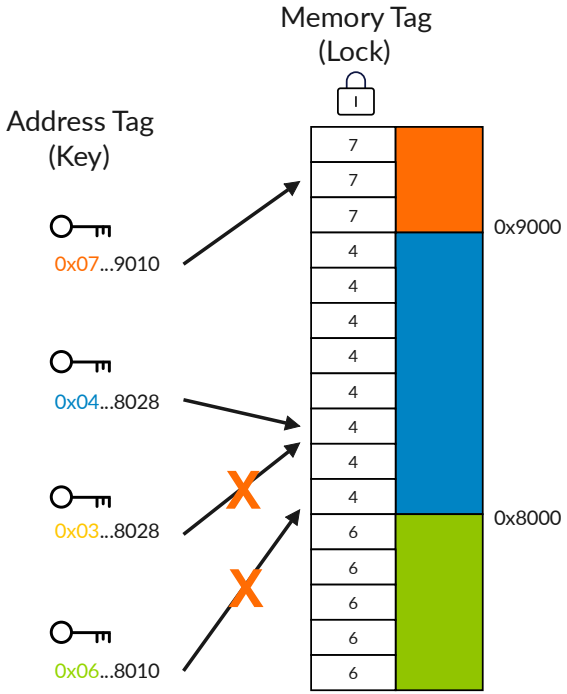
April 21, 2026

Andrew Kwong

THE UNIVERSITY
*of* NORTH CAROLINA
*at* CHAPEL HILL

Slides adapted from Mengjia Yan (shd.mit.edu)

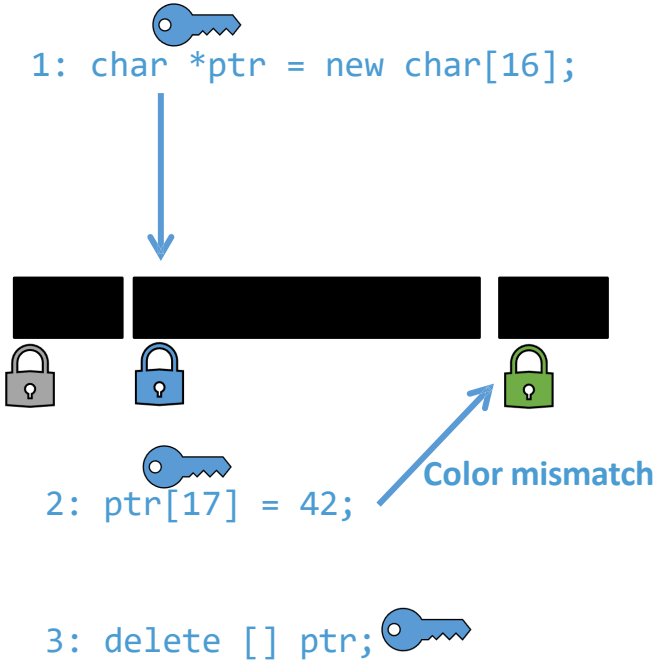
# ARM MTE (Memory Tagging Extension)




- The concept of keys and locks
- Memory locations are tagged by adding four bits of metadata to each 16 bytes of physical memory

*Armv8.5-A Memory Tagging Extension White paper*
*https://security.googleblog.com/2019/08/adopting-arm-memory-tagging-extension.html*

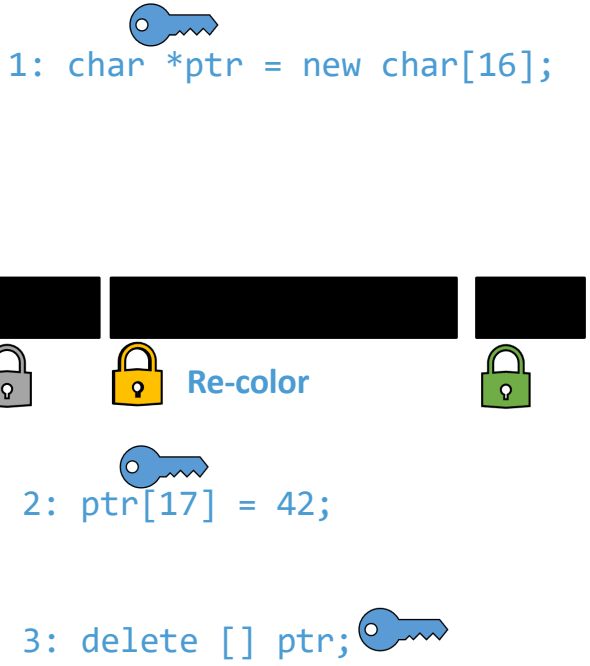
# ARM MTE (Memory Tagging Extension)




- The concept of keys and locks
- Memory locations are tagged by adding four bits of metadata to each 16 bytes of physical memory

*Armv8.5-A Memory Tagging Extension White paper*
*https://security.googleblog.com/2019/08/adopting-arm-memory-tagging-extension.html*

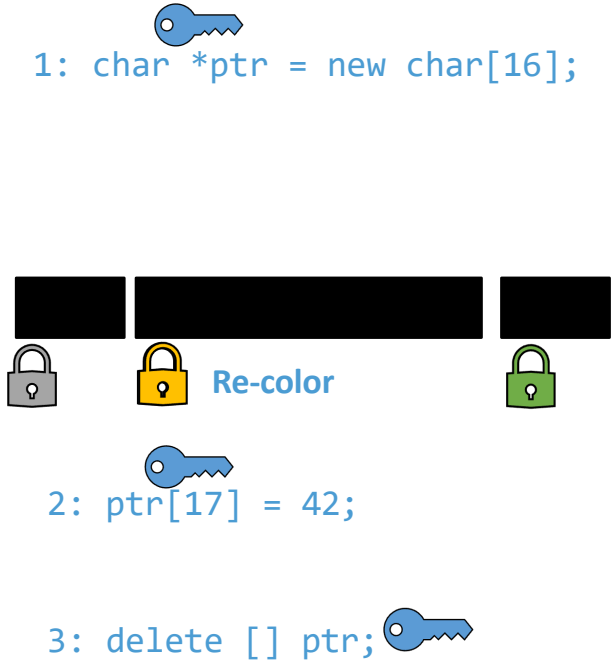
# ARM MTE (Memory Tagging Extension)




- The concept of keys and locks
- Memory locations are tagged by adding four bits of metadata to each 16 bytes of physical memory

*Armv8.5-A Memory Tagging Extension White paper*
*https://security.googleblog.com/2019/08/adopting-arm-memory-tagging-extension.html*

# Analysis of ARM MTE

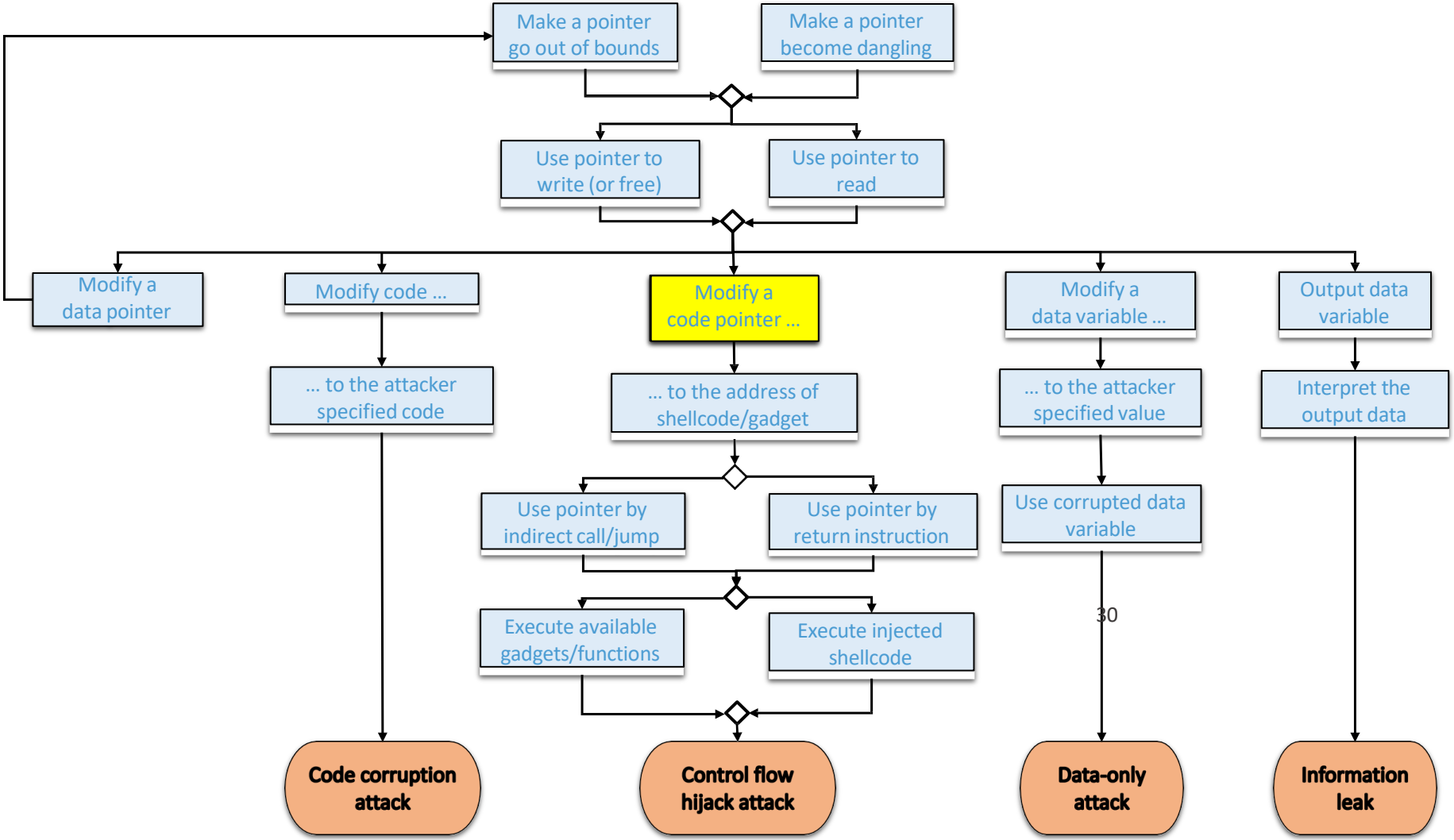



- Where to store tags (key and lock)?
  - Pointer tag is stored in top 4 unused bits inside the pointer (no extra register needed)
  - Physical memory tag is stored in hardware (new hardware needed for both DRAM and cache)
- Limited tag bits
  - Cannot ensure two allocations have different colors
  - But can ensure that the tags of sequential allocations are always different

*Armv8.5-A Memory Tagging Extension White paper*
*https://security.googleblog.com/2019/08/adopting-arm-memory-tagging-extension.html*

# Analysis of ARM MTE

- Security:
- - Coarse-grained spatial safety. Non-sequential violation is detected probabilistically
- + Can support temporal safety similar to spatial safety
- Performance and other overhead:
- + Storage overhead is ok
  - 4 bits per 16 bytes
- + Performance overhead is low, mostly lies in the allocation and free time, since need to modify tags in bulk
- Compatibility:
- + To protect heap, modify libraries to do malloc and free; modify OS to trap on invalid pointer. No extensive code rewritten needed.


Make a pointer go out of bounds
Make a pointer become dangling
Use pointer to write (or free)
Use pointer to read
Modify a data pointer
Modify code …
Modify a code pointer …
Modify a data variable …
Output data variable
… to the attacker specified code
… to the address of shellcode/gadget
… to the attacker specified value
Interpret the output data
Use pointer by indirect call/jump
Use pointer by return instruction
Use corrupted data variable
Execute available gadgets/functions
Execute injected shellcode
Code corruption attack
Control flow hijack attack
Data-only attack
Information leak

# Control-flow Integrity

- To maintain code pointer integrity
- Naive idea:
  - Make pointer immutable (read-only)
    - Only work for global offset table
- How about other pointers?
  - Return address?
  - Programmer-defined function pointers

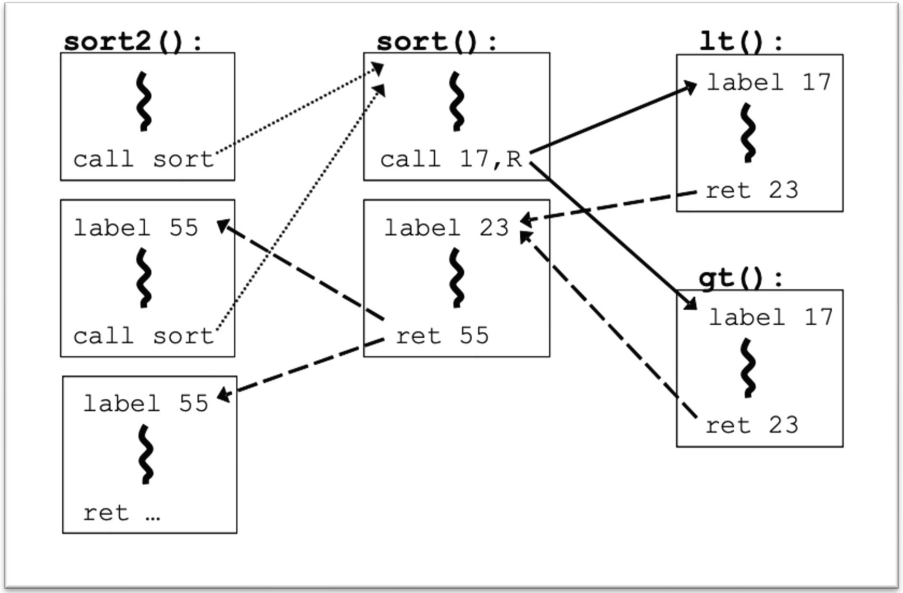
# Control Flow Integrity (CFI)

```
bool lt(int x, int y) {
    return x < y;
}

bool gt(int x, int y) {
    return x > y;
}

sort2(int a[], int b[], int len)
{
    sort( a, len, lt );
    sort( b, len, gt );
}

sort(int x[], int len, fun_ptr)
{
    for(int i=0; ….)
        for (int j=i; ….)
            if (fun_ptr(x[i], x[j]))
                … //swap x[i] and x[j]
}
```



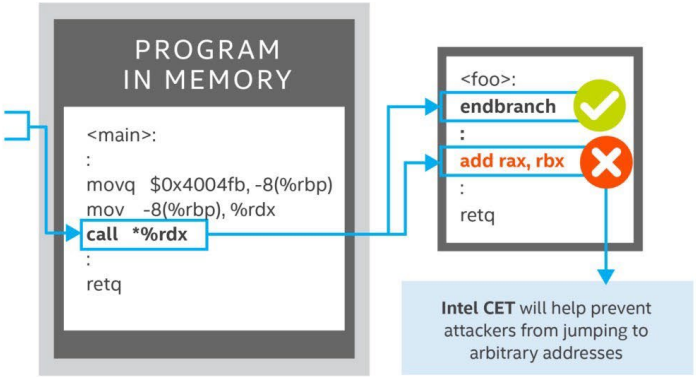

*Control-Flow Integrity Principles, Implementations, and Applications; Mart´ın Abadi, et al. CCS'05*

# Intel® Control-Flow Enforcement Technology (Intel CET)

INTEL CET = INDIRECT BRANCH TRACKING (IBT) + SHADOW STACK (SS)

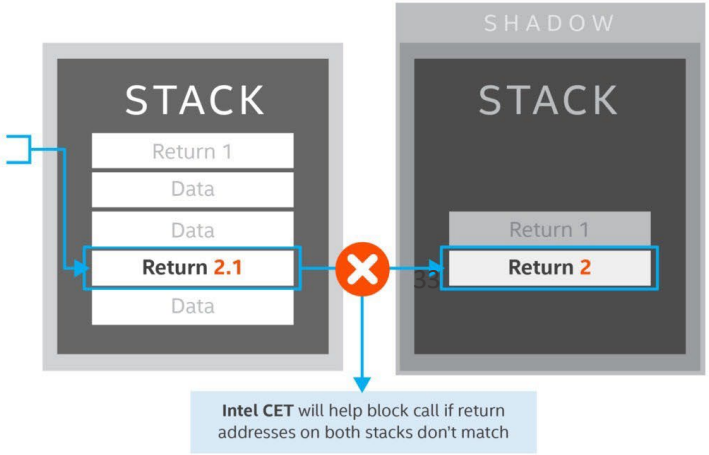
## INDIRECT BRANCH TRACKING (IBT)

IBT delivers indirect branch protection to defend against jump/call oriented programming (JOP/COP) attack methods.



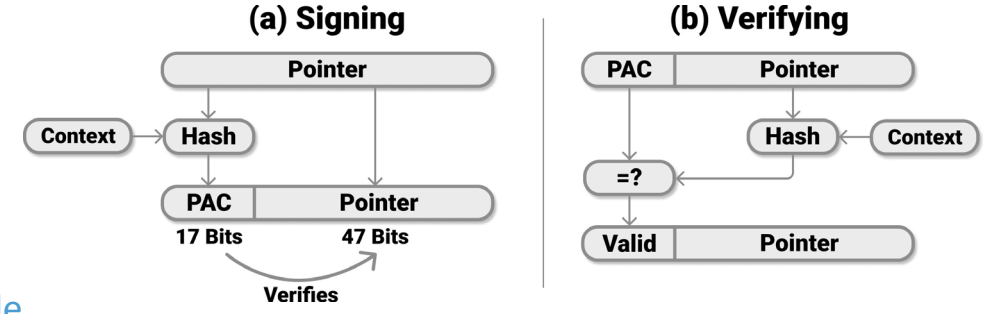

## SHADOW STACK (SS)

SS delivers return address protection to defend against return-oriented programming (ROP) attack methods.



# ARM PA (Pointer Authentication)

- Introduced in 2017, widely used in Apple processors
- 64-bit pointer, but 48-bit virtual address space
  - Unused high bits
- Hash:
  - A tweakable message authentication code (MAC)
  - ARM calls it **PAC (pointer authentication code)**
- Context:
  - secret key
  - salt (could be the stack pointer)




Function prologue

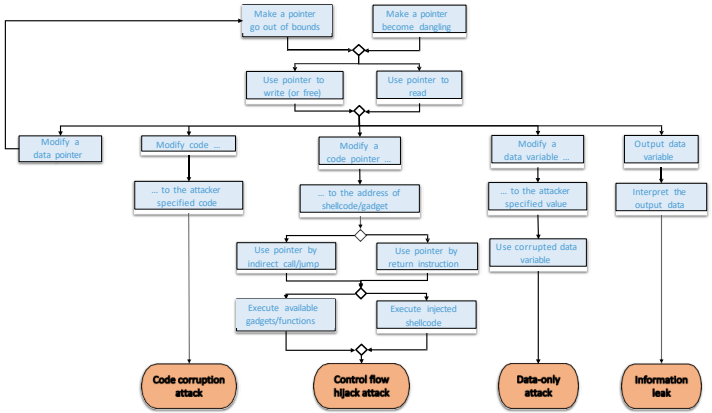
```
1  pacia lr, sp
2  sub   sp, sp, #0x40
3  str   lr, [sp, #0x30]
4  ...
```

Function epilogue

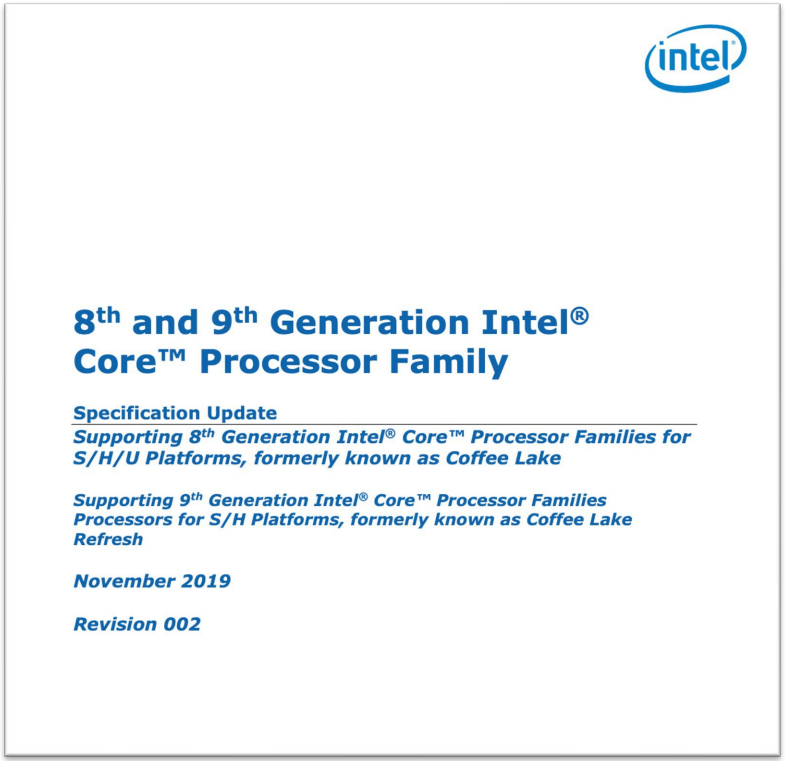
```
1  ldr   lr, [sp, #0x30]
2  add   sp, sp, #0x40
3  autia lr, sp
4  ret
```

# Summary

- Memory corruption problems: An eternal war
- Attack variations and mitigations
- Trade-offs in hardware support widely used on real systems



# What is Errata?




It is a compilation of device and document errata and specification clarifications and changes, which is intended for hardware system manufacturers and for software developers of applications, operating system, and tools.

**Errata** are design defects or errors. Errata may cause the processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given stepping must assume that all errata documented for that stepping are present on all devices.

https://www.intel.com/content/www/us/en/products/docs/processors/core/8th-gen-core-spec-update.html

# Errata Table Example

## 3.2 Errata Summary Information

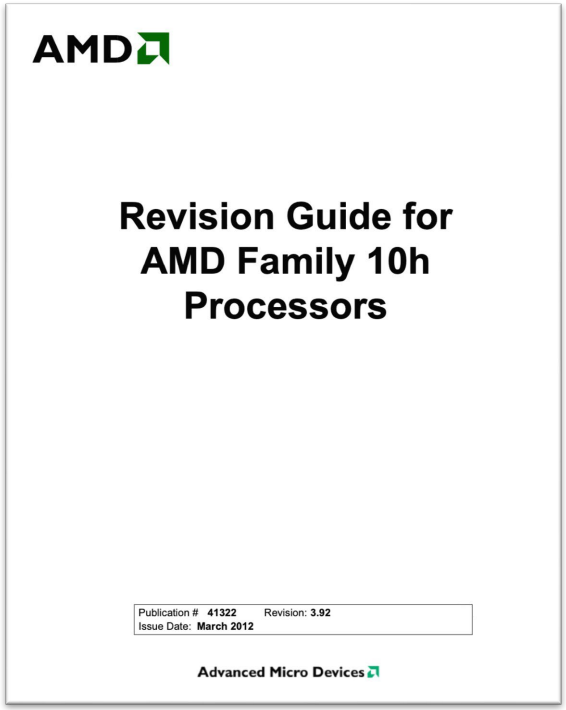
**Table 4-3. Errata Summary Table**

| ID | Processor Line / Stepping | | | | | | | Title |
|---|---|---|---|---|---|---|---|---|
| | S | | | | H | | U | |
| | B0 42 | U0 62 | P0 82 | R0 82 | U0 62 | R0 82 | D0 43e | |
| 001 | No Fix | No Fix | No Fix | I | | | | Reported Memory Type May Not Be Used to |
| 002 | No Fix | No Fix | No Fix | I | | | | |
| 003 | No Fix | No Fix | No Fix | I | | | | |
| 004 | No Fix | No Fix | No Fix | I | | | | |

| | |
|---|---|
| **001** | **Reported Memory Type May Not Be Used to Access the VMCS and Referenced Data Structures** |
| **Problem** | Bits 53:50 of the IA32_VMX_BASIC MSR report the memory type that the processor uses to access the VMCS and data structures referenced by pointers in the VMCS. Due to this erratum, a VMX access to the VMCS or referenced data structures will instead use the memory type that the memory-type range registers (MTRRs) specify for the physical address of the access. |
| **Implication** | Bits 53:50 of the IA32_VMX_BASIC MSR report that the write-back (WB) memory type will be used, but the processor may use a different memory type. |
| **Workaround** | Software should ensure that the VMCS and referenced data structures are located at physical addresses that are mapped to WB memory type by the MTRRs. |
| **Status** | For the steppings affected, refer the Summary Table of Changes. |

# More Errata

Occasionally, AMD identifies product errata that cause the processor to deviate from published specifications. Descriptions of identified product errata are designed to assist system and software designers in using the processors described in this revision guide. This revision guide may be updated periodically.



AMD

Revision Guide for AMD Family 10h Processors

Publication # 41322 Revision: 3.92
Issue Date: March 2012

Advanced Micro Devices

### 298 L2 Eviction May Occur During Processor Operation To Set Accessed or Dirty Bit

**Description**

The processor operation to change the accessed or dirty bits of a page translation table entry in the L2 from 0b to 1b may not be atomic. A small window of time exists where other cached operations may cause the stale page translation table entry to be installed in the L3 before the modified copy is returned to the L2.

In addition, if a probe for this cache line occurs during this window of time, the processor may not set the accessed or dirty bit and may corrupt data for an unrelated cached operation.

**Potential Effect on System**

One or more of the following events may occur:

- Machine check for an L3 protocol error. The MC4 status register (MSR0000_0410) is B2000000_000B0C0Fh or BA000000_000B0C0Fh. The MC4 address register (MSR0000_0412) is 26h.
- Loss of coherency on a cache line containing a page translation table entry.
- Data corruption.

**Suggested Workaround**

BIOS should set MSRC001_0015[3] (HWCR[TlbCacheDis]) to 1b and MSRC001_1023[1] to 1b.

In a multiprocessor platform, the workaround above should be applied to all processors regardless of

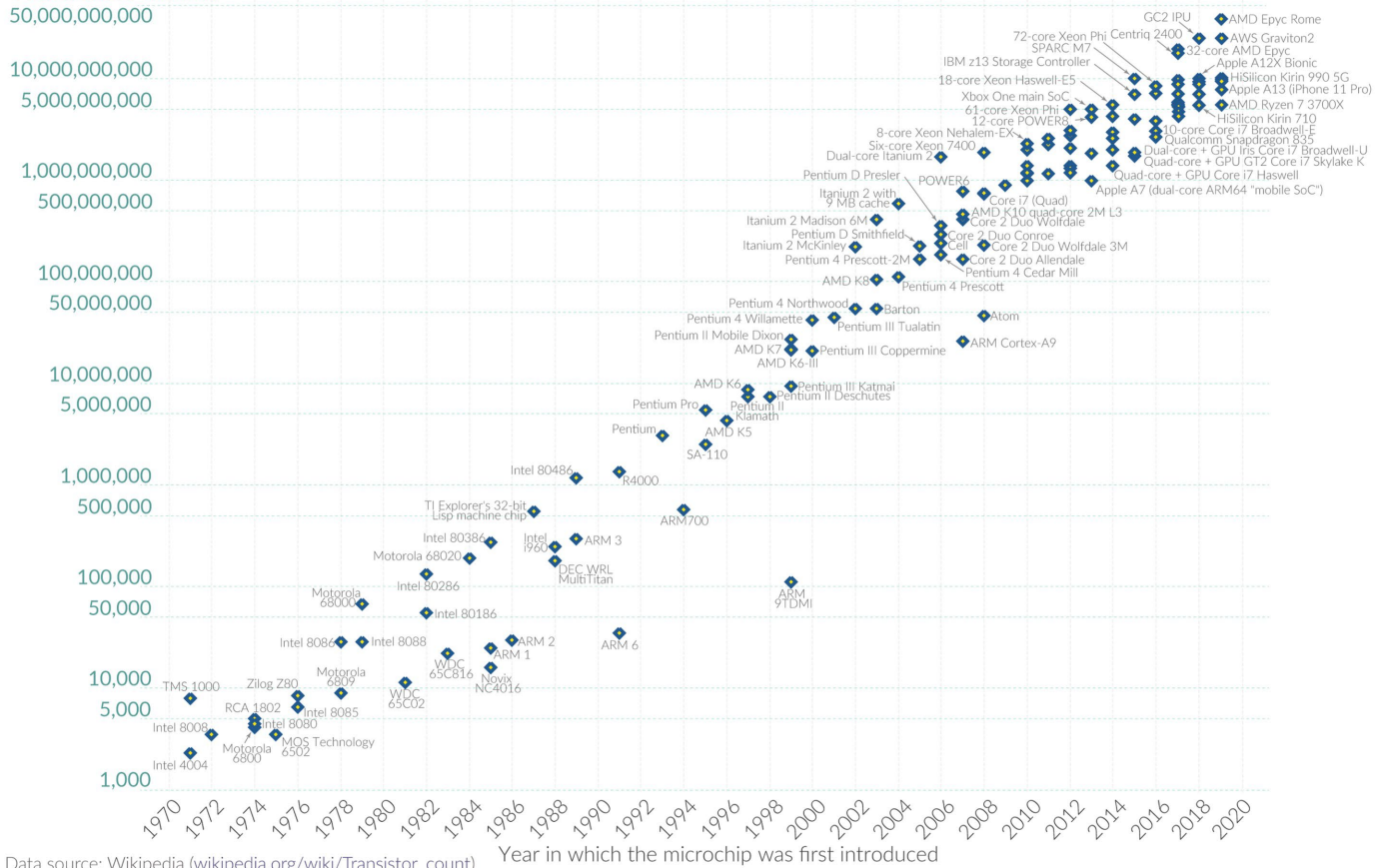
https://www.amd.com/system/files/TechDocs/41322_10h_

10,000 x

# Moore's Law: The number of transistors on microchips doubles every two years

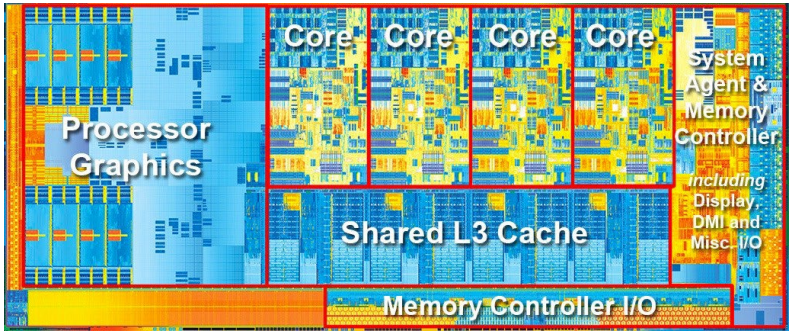
Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years.
This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.




Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world's largest problems.

# Errata Statistics




The Core-i7 processor with integrated graphics card in 2012 with 1,400M Transistors

**4 years; 136 errata; 3 bugs/month**

A **4-fold increase** in bugs in Intel processor designs **per generation**. Approximately 8000 bugs designed into the Pentium 4 ('Willamette')

from *https://www.cl.cam.ac.uk/~jrh13/slides/nijmegen-21jun02/slides.pdf*

*SPECS: A Lightweight Runtime Mechanism for Protecting Software from Security-Critical Processor Bugs; Hicks et al; ASPLOS'15*
*https://github.com/impedimentToProgress/specs*

# Outline

- Hardware Bug Examples
  - What do they look like? The discovery process? Impact?
  - #1: The famous Pentium FDIV bug
  - #2: SYSRET 64-bit OS privilege escalation vulnerability on Intel CPU
  - #3: Branch history injection attack
- How to discover hardware bugs?
  - Manual efforts, testing
  - Fuzzing

# Bug #1: Pentium FDIV Bug

- What is the specification for floating-point computation?
  - Floating is encoded as $(1 + f) \times 2^e, 0 \leq f < 1, e \in Z$
  - Example: $1/10 = 1.9999 \ldots 9a \times 2^{-4}$ (in hexadecimal)
  - We always have errors when doing floating-point computation, because we have limited number of bits for each floating number
- The specification allows error to occur after bit $x$

| | Single precision | Double precision | Extended precision |
|---|---|---|---|
| Word size in bits | 32 | 64 | 80 |
| Bits for $f$ | 23 | 52 | 63 |
| Bits for $e$ | 8 | 11 | 15 |
| Relative accuracy | $2^{-23} \approx 1.2 \cdot 10^{-7}$ | $2^{-52} \approx 2.2 \cdot 10^{-16}$ | $2^{-63} \approx 1.1 \cdot 10^{-19}$ |
| Approximate range | $2^{\pm 127} \approx 10^{\pm 38}$ | $2^{\pm 1023} \approx 10^{\pm 308}$ | $2^{\pm 16383} \approx 10^{\pm 4964}$ |

The Pentium FDIV bug: see errors much earlier than the expected $x$ bits

# The Discovery Process #1: Nicely's Prime

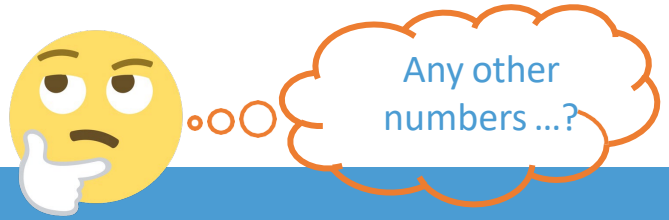
- Thomas Nicely, a mathematics professor, tried to compute reciprocal of prime numbers: $p = 824,633,702,441$
- The correct result:

$$1/p = 1.212659629408667 \times 10^{-12}$$

- But the new Pentium processor gave:

$$1/p = 1.212659624891158 \times 10^{-12}$$

- Took him four months to confirm the problem was NOT in his program -> math libraries -> compilers -> operating system, but in the hardware

# The Discovery Process #2: Kaiser's List

- Andreas Kaiser, a computer consultant
  - Generate $25\ billion$ random integers and checked the accuracy of the computed reciprocals. $23$ are incorrect.

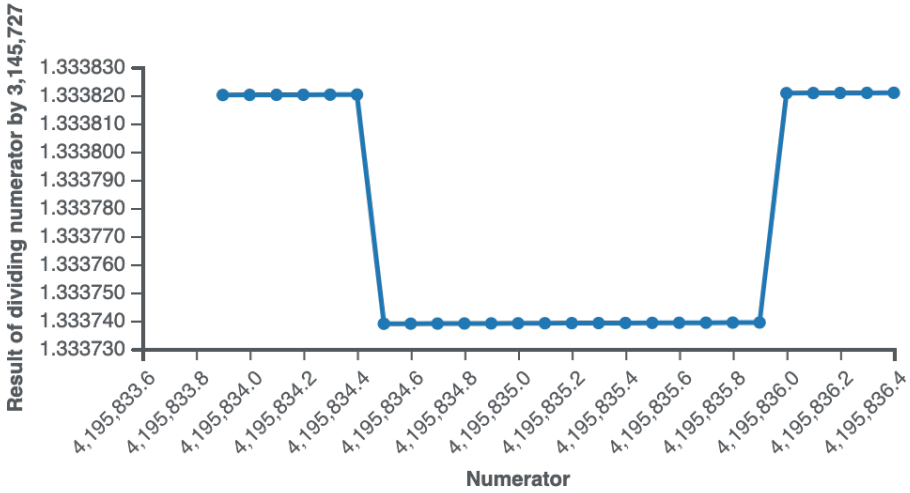
| | | |
|---|---|---|
| 3221224323 | = | $1.7ffff70600000 \cdot 2^{31}$ |
| 12884897291 | = | $1.7ffff70580000 \cdot 2^{33}$ |
| 206158356633 | = | $1.7ffff704c8000 \cdot 2^{37}$ |
| 824633702441 | = | $1.7fffff7052000 \cdot 2^{39}$ |
| 1443107810341 | = | $1.4fffedac25000 \cdot 2^{40}$ |
| 6597069619549 | = | $1.7fffff7057400 \cdot 2^{42}$ |
| 9895574626641 | = | $1.1fffc6bc2a200 \cdot 2^{43}$ |
| 13194134824767 | = | $1.7ffff704e7e00 \cdot 2^{43}$ |
| 26388269649885 | = | $1.7ffff704fdd00 \cdot 2^{44}$ |
| 52776539295213 | = | $1.7ffff7046f680 \cdot 2^{45}$ |

Patterns?

- Many start with $1.7fffff$
- In another word, the first 20 bits after the leading bit have to be a single zero, followed by at least 19 ones

# The Discovery Process #3: Coe's Ratio

- Tim Coe, electrical engineer, had designed floating-point chips
  - $\frac{4,195,835}{3,145,727} = 1.33382044\ldots$ (correct) $1.33373906\ldots$ (Pentium)

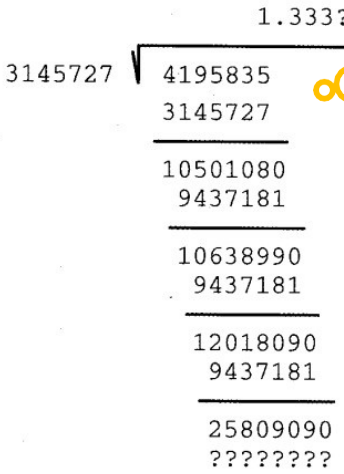



Differ after the 4th digit

The errors involve $y/x$ where $x$ and $y$'s bit patterns conspire to excite the bug at an early stage in the division.

# Bug Explanation: FDIV

- Shift-and-subtract




How do we choose the quotient as a human being?

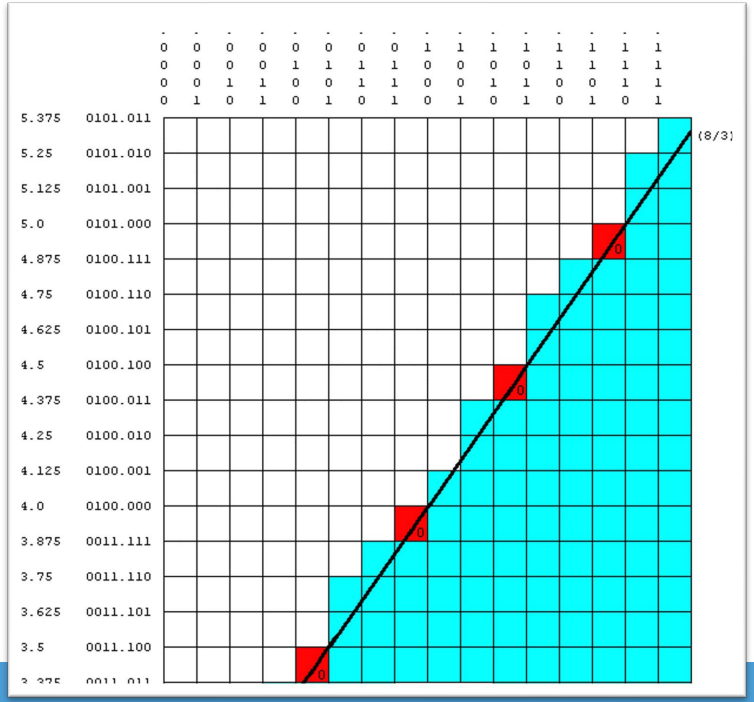
A combination of trial and error, experience, pattern matching and luck.

- Fast algorithm: Sweeney, Robertson, and Tocher (SRT) algorithm Radix-4:
  - Choose quotient from $0, +1, -1, +2, -2$;
  - If the current quotient is incorrectly chosen, we can recover it from the next iteration
  - Guess the quotient based on the first few digits => use a 2D table to lookup

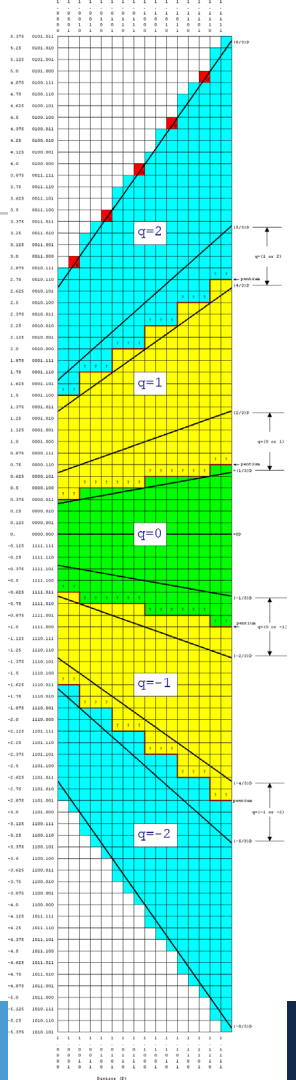
# Bug Explanation: SRT Table

first 5 bits of the divisor

first 7 bits of the remainder




- 2048 cells in total
- **5 cells are not initialized**
- When will the bug be triggered?



## How Frequently the bug can be triggered?

- Intel: an average spreadsheet user could encounter this flaw once in every 27,000 years, assuming 1,000 divisions per day.
- IBM: suspended sales of Pentium-based models and said it is as many as 20 mistakes per day.
- Who actually got affected?
  - Normal users?
  - Post quantum crypto?
  - Wall street? Financial prediction programs? Did the Pentium bug flip a trading decision from buy to hold to sell?
  - Difficult to calibrate

# Consequences/Impacts

- Intel's bad responses
  - Conditional replacement (customers need to claim they do get influenced by the bug) ➔ disastrous press
  - No-questions-asked replacement ➔ $475M cost in 1994, 10% replacements
- Potential long-term impact:
  - Random test may not be a good idea. Exhaustive testing has a scalability problem.
  - A marked increase in the use of formal verification and number theory in hardware design

**Some humor for you:**

**Q: How many Pentium designers does it take to screw in a light bulb?**
**A: 1.99904274017, but that's close enough for non-technical people.**

**Q: What do you get when you cross a Pentium PC with a research grant?**
**A: A mad scientist.**

**Do you think it bothers x86 users that the 486 is a functional upgrade to the Pentium?**

**In response to the Pentium bug, PowerMac officials have announced that they will be adding the control panel "Pentium Switcher" that allows users to decide whether the PowerMac should emulate pre-Pentium or post-Pentium FDIV behaviour.**

**TOP TEN NEW INTEL SLOGANS FOR THE PENTIUM**
**----------**

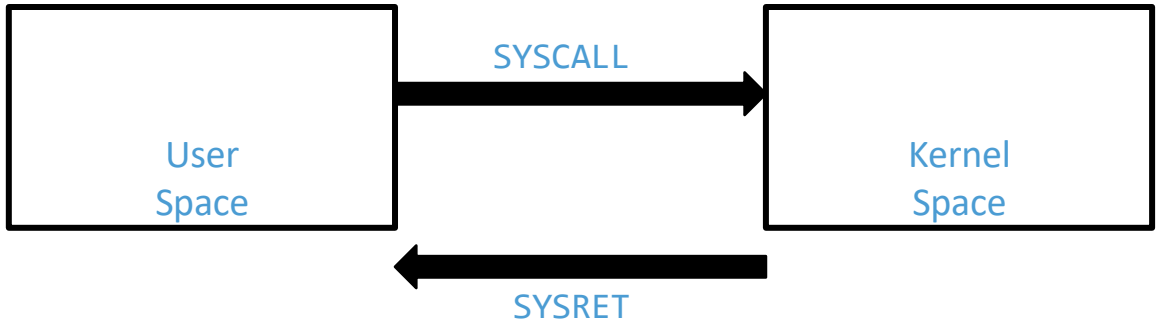
**9.9999973251 It's a FLAW, Dammit, not a Bug**
**8.9999163362 It's Close Enough, We Say So**
**7.9999414610 Nearly 300 Correct Opcodes**
**6.9999831538 You Don't Need to Know What's Inside**
**5.9999835137 Redefining the PC--and Mathematics As Well**
**4.9999999021 We Fixed It, Really**
**3.9998245917 Division Considered Harmful**
**2.9991523619 Why Do You Think They Call It *Floating* Point?**
**1.9999103517 We're Looking for a Few Good Flaws**
**0.9999999998 The Errata Inside**

http://davefaq.com/Opinions/Stupid/Pentium.html#glitch

# Bug #2: A SYSRET Bug

64-bit x86 instruction set: AMD64, Intel 64

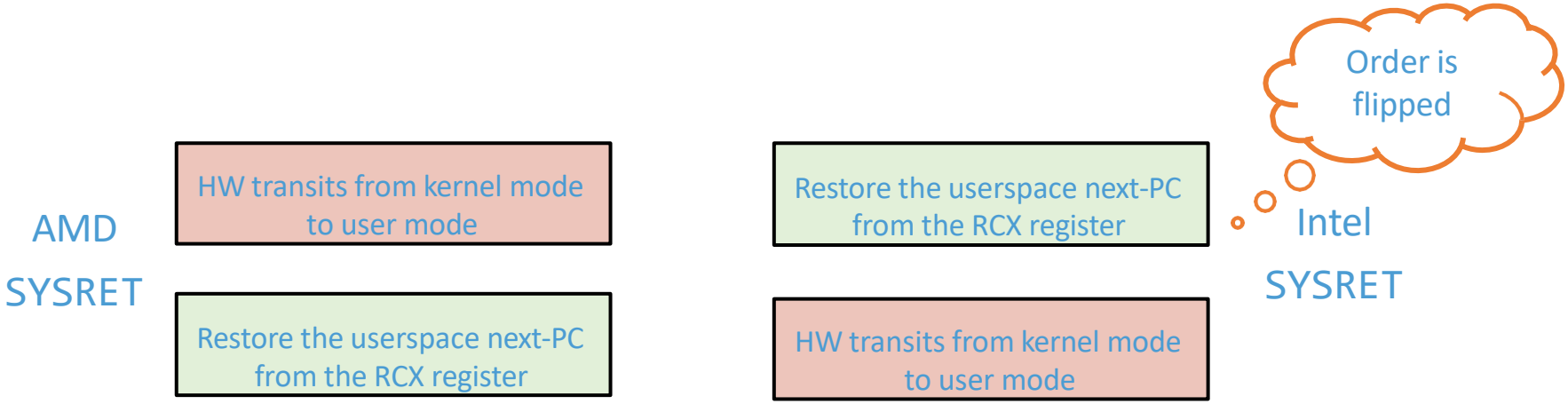
SYSCALL
- HW transits from user mode to kernel mode
- Save the userspace next-PC to the RCX register
- Jump to a kernel syscall entry point



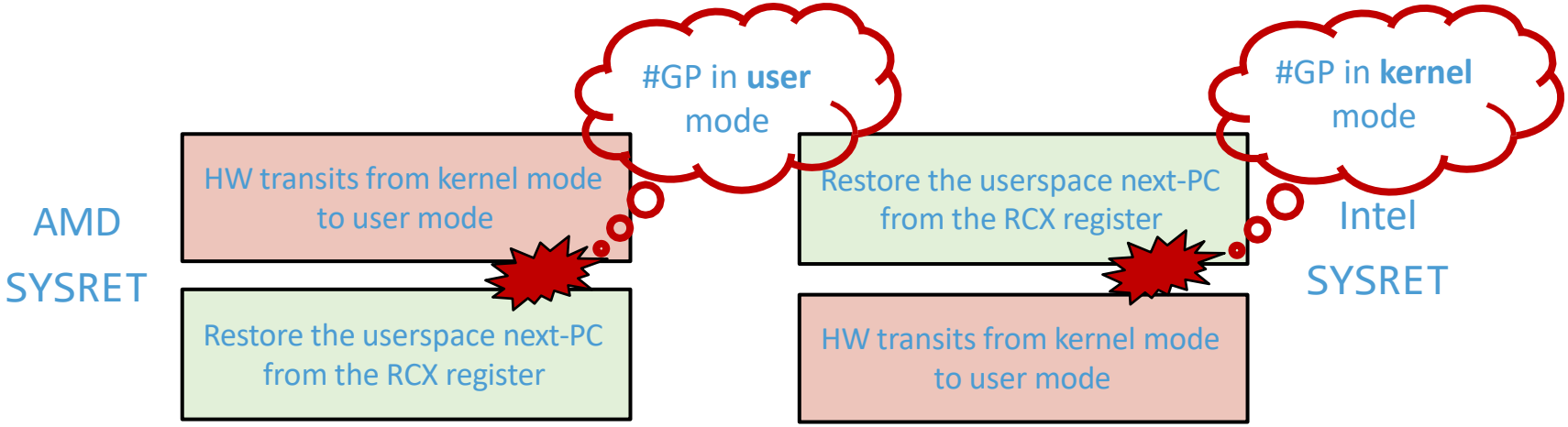

SYSRET
- HW transits from kernel mode to user mode
- Restore the userspace next-PC from the RCX register

*A Stitch In Time Saves Nine: A Stitch In Time Saves Nine: A Case Of Multiple OS Vulnerability; Rafal Wojtczuk; BlackHat, 2012 Model Checking to Find Vulnerabilities in an Instruction Set Architecture; Bradfield et al; HOST'16*

# Two Different Specifications for SYSRET



# SYSRET Vulnerability




If RCX holds a non-canonical address, the SYSRET will generates a #GP (general protection fault)
Canonical means that given 48-bit virtual address space, the high 16 bits (bits 63-48) of a virtual address have same value as bit 47.

# How SYSRET is used in kernel code?

- What do we do before we transition from kernelspace to userspace?

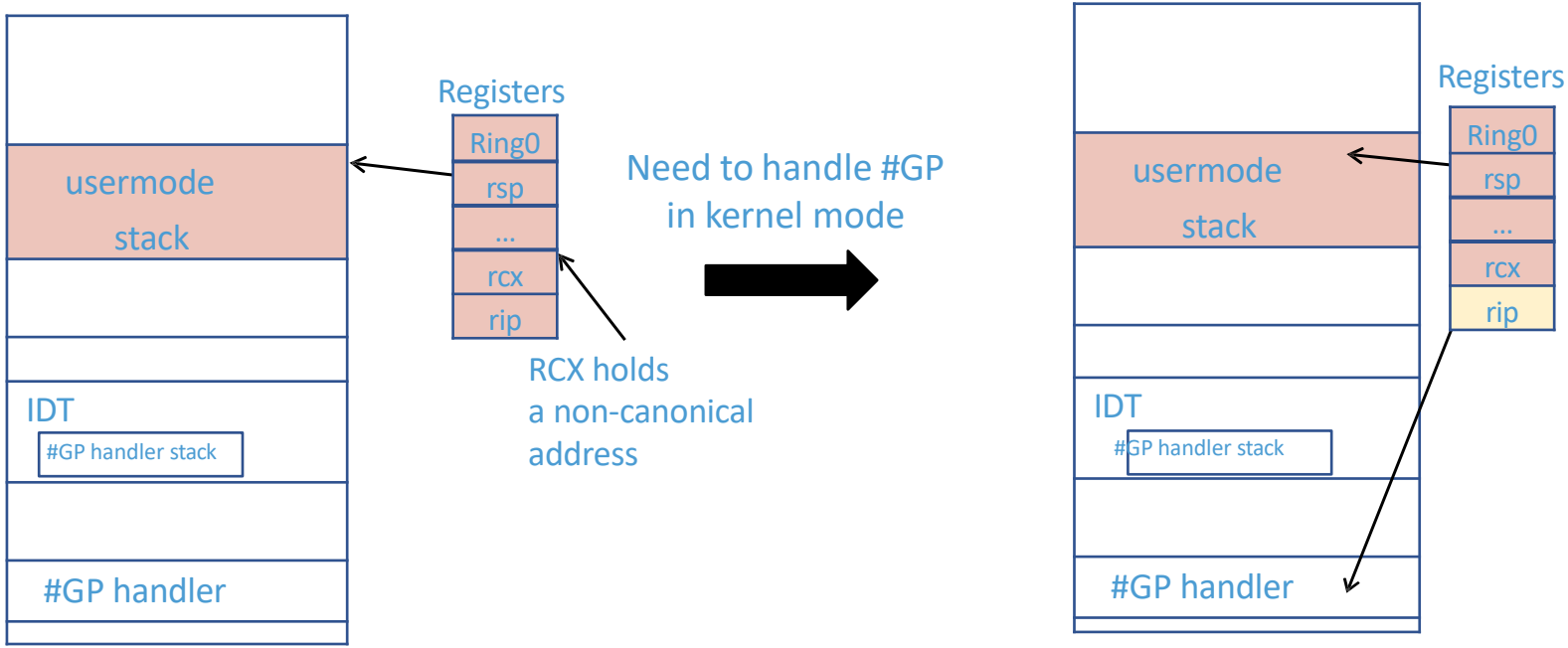
At this point, all the registers are user-controlled (attacker-controlled)

```
movq RCX(%rsp), %rcx
movq RIP(%rsp), %r11
cmpq %rcx, %r11 /* SYSRET requires RCX == RIP */
jne swapgs_restore_regs_and_return_to_usermode

...
/* populate all the registers using data from userstack */
sysret
```

https://elixir.bootlin.com/linux/latest/source/arch/x86/entry/entry_64.S

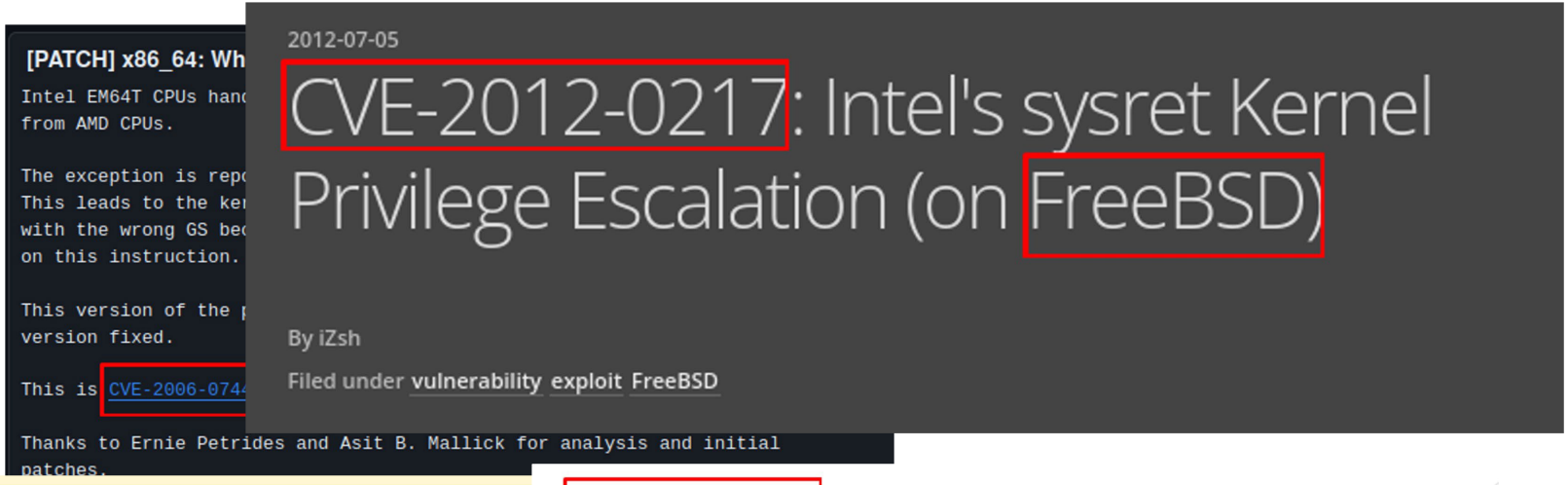
# SYSRET Attack on Intel Processors




Before executing SYSRET, all registers have been restored using usermode context

Assume rip points to kernel stack and start using it --> can **overwrite kernel data**

# Longtime Intel x86 OS Bug




## CVE-2012-0217 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting rea
changes to the information provided.

### Description

The x86-64 kernel system-call functionality in Xen 4.1.2 and earlier, as used in Citrix XenServe
Oracle Solaris 11 and earlier; illumos before r13724; Joyent SmartOS before 20120614T184600Z; FreeBSD before 9.0-RELEASE-p3; NetBSD 6.0 Beta and earlier; Microsoft Windows Server 2008 R2 and R2 SP1 and Windows 7 Gold and SP1; and possibly other operating systems, when running on an Intel processor, incorrectly uses the sysret path in cases where a certain address is not a

## CVE-2014-4699: Linux Kernel ptrace/sysret vulnerability analysis

by Vitaly Nikolenko

Posted on July 21, 2014 at 6:52PM

# It’s not a bug, it’s a feature

## Description

SYSRET is a companion in … tem-call handler to user code at privilege level 3. It … 1.[1] With a 64-bit operand size, SYSRET remains in 6… e low 32 bits of the regis-ters are loaded.

## Operation

```
IF (CS.L ≠ 1 ) or (IA32_EFER.LMA ≠ 1) or (IA32_EFER.SCE ≠ 1)
(* Not in 64-Bit Mode or SYSCALL/SYSRET not enabled in IA32_EFER *)
    THEN #UD; FI;
IF (CPL ≠ 0) THEN #GP(0); FI;

IF (operand size is 64-bit)
    THEN (* Return to 64-Bit Mode *)
        IF (RCX is not canonical) THEN #GP(0);
        RIP := RCX;
    ELSE (* Return to Compatibility Mode *)
        RIP := ECX;
FI;
RFLAGS := (R11 & 3C7FD7H) | 2;              (* Clear RF, VM, reserved bits; set bit 1 *)

IF (operand size is 64-bit)
    THEN CS.Selector := IA32_STAR[63:48]+16;
    ELSE CS.Selector := IA32_STAR[63:48];
FI;
CS.Selector := CS.Selector OR 3;            (* RPL forced to 3 *)
(* Set rest of CS to a fixed value *)
CS.Base := 0;                               (* Flat segment *)
CS.Limit := FFFFFH;                         (* With 4-KByte granularity, implies a 4-GByte limit *)
CS.Type := 11;                              (* Execute/read code, accessed *)
CS.S := 1;
CS.DPL := 3;
CS.P := 1;
IF (operand size is 64-bit)
    THEN (* Return to 64-Bit Mode *)
```

# It’s not a bug, it’s a feature

Intel claims that this vulnerability is a software implementation issue, as their processors are functioning as per their documented specifications. However, software that fails to take the Intel-specific SYSRET behavior into account may be vulnerable.

https://www.kb.cert.org/vuls/id/649219

## Who to blame?

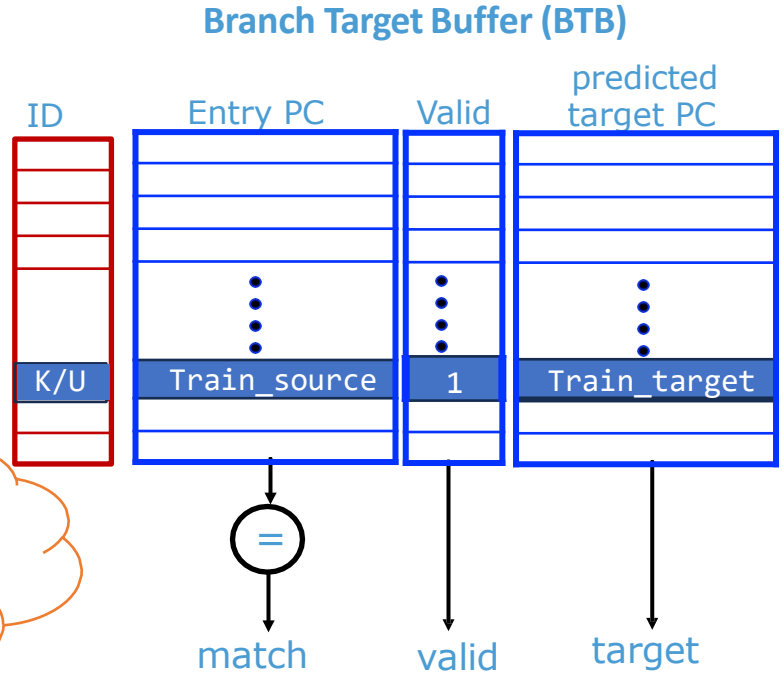
- Intel claims it is not an errata
  - *Errata are design defects or errors that may cause ... behavior to deviate from published specifications*.
  - This behavior is consistent with Intel's specification
  - So the problem is the specification is incorrect
- Intel SDM (software development manual) 3400 pages. We cannot assume the specification is always correct.
- Research question: how can we know the ISA specification is correct?
  - Some research efforts to verify ISA specification

# Bug #3: eIBRS Vulnerability

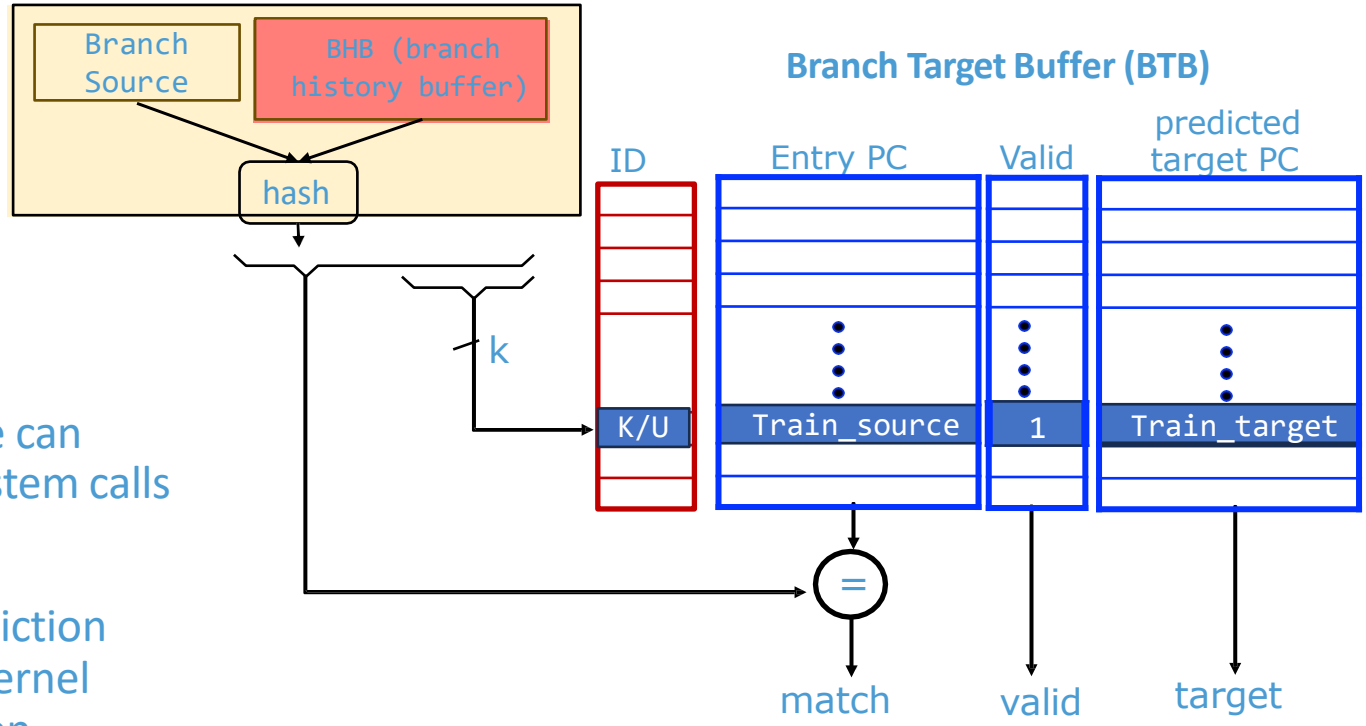
- Recap Spectre v2
- eIBRS: Enhanced Indirect Branch Restricted Speculation. Advertised as a mitigation against Spectre v2.

**Specification:**
Do not let lower-privileged code interfere with the branch prediction target of the high-privilege code.
OR
Isolate BTB entries across privilege levels.

What does this mean?
Non-interference?
A vague specification.



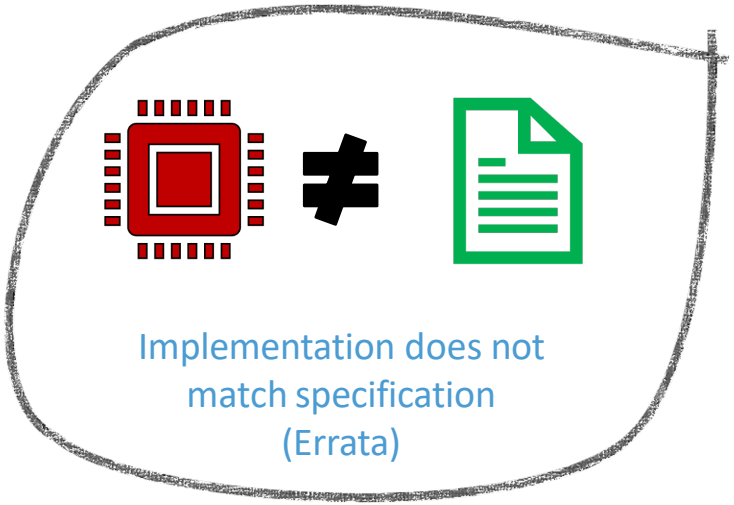
# Recap the Problem: Branch History Injection




- #1: Userspace code can trigger different system calls
- #2: Userspace prediction history can affect kernel space BTB prediction

*Barberis et al. Branch History Injection: On the Effectiveness of Hardware Mitigations Against Cross-Privilege Spectre-v2 Attacks. USENIX'22 https://www.vusec.net/projects/bhi-spectre-bhb/*

# Summary



Implementation does not match specification (Errata)

Bugs in the specification

Vague specification

- Next: How to find hardware bugs?
  - Get ideas from the software

# Software Bugs Hunting/Fixing

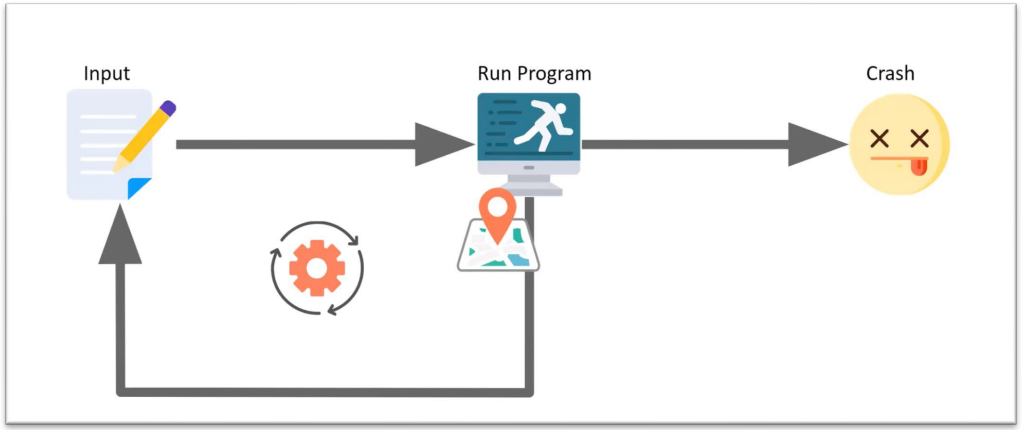
- Approach 1: Manual effort
  - Hire a lot of experts and stare at the code
  - Regression test ➔ but need to be updated
- Approach 2: How about randomly generating test cases?
  - Fuzzing

# Fuzzing

# Fuzzing In A Nutshell

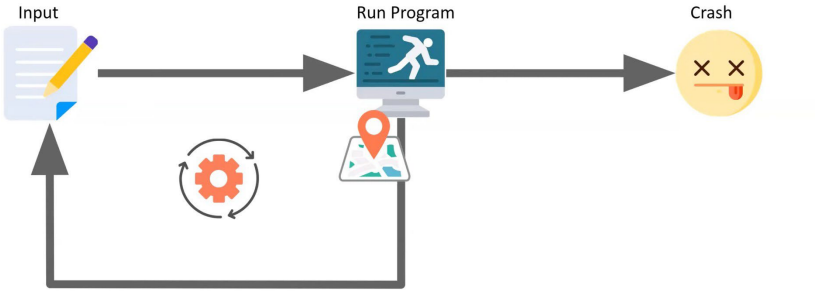
- Automatically generate test examples
- 1999, Alan Cox at University of Wales discovered a vulnerability in Linux kernel by simply running a proram generating random input and feed into the kernel

- Simple yet effective
- Industry standard




*From Riding the Fuzzing Hype Train (RAID'21 Keynote)*

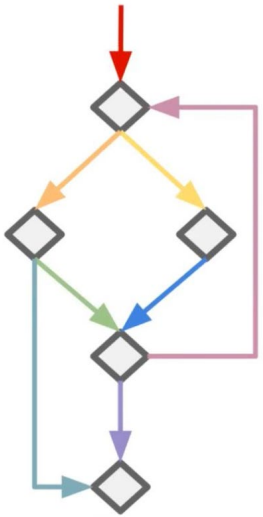
# Fuzzing Components

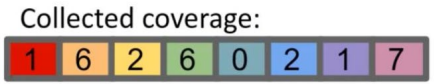



- Random seeds
  - Sometimes need formatted inputs, e.g., PDF reader
- A criteria to check whether the outcome is as expected or not.
  - Specification
  - Assertions
- Heuristics for generating new tests => feedback loop for better efficiency

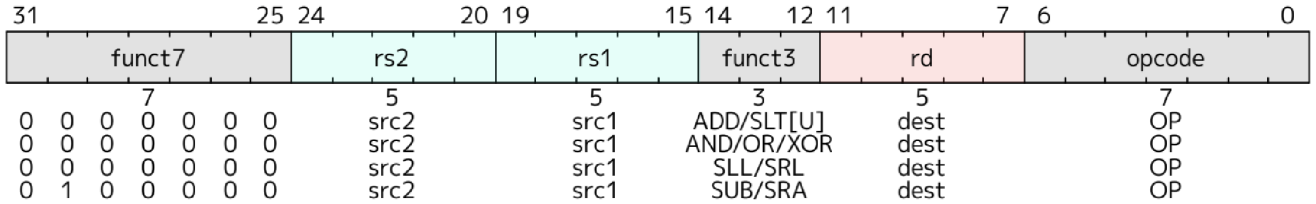
# Types of Fuzzing

- Blackbox
- Whitebox
- Greybox





# Example: Hidden Instructions



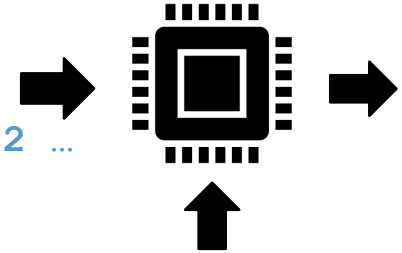

- Hidden instructions: secret instructions that give backdoor or powerful access to processor internals
- Secret processor functionality: Appendix H
- An example:
  - Pentium F00F bug, an invalid instruction freezes the cpu, discovered in 1997
  - A Ring 3 process can DOS (denial of service) a processor
  - The invalid instruction encoding is: **`F0 0F C7 [C8-CF]`**

*Breaking the x86 ISA, Christopher Domas; Blackhat'17*

# Search for Hidden Instructions

Instructions:

```
0F 6A 60 6A 79 6D C6 02 …
```

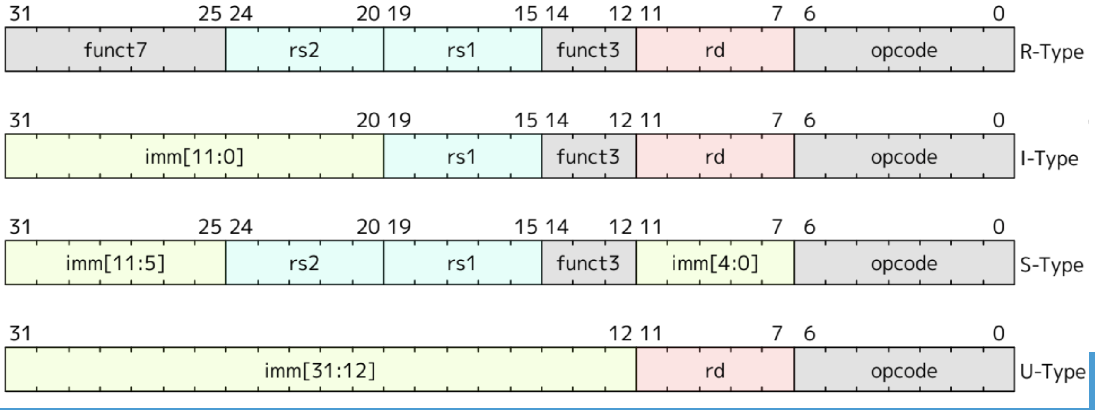


ISA specification:

Valid instructions (in spec)

Invalid instructions
(#UD exception, invalid opcode)

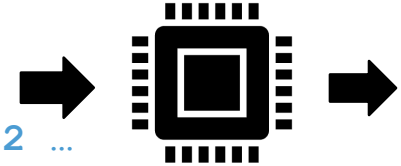
Hidden instructions (not in spec,
but can execute, no #UD exception)



# Challenges #1: Detect Hidden Instruction

Instructions:

```
0F 6A 60 6A 79 6D C6 02 …
```



Valid instructions (in spec)

Invalid instructions
(#UD exception, invalid opcode)

**Hidden instructions (not in spec, but can execute, no #UD exception)**

How to capture this case?

# SandSifter and Findings


- Hidden instructions across Intel and AMD processors
- Hardware errata, something like FOOF

# More Hardware Fuzzing Examples

- Zenbleed: found a CPU bug via post-silicon fuzzing

```
movnti [rbp+0x0],ebx
rcr dh,1
sub r10, rax
rol rbx, cl
xor edi,[rbp-0x57]
```

```
movnti [rbp+0x0],ebx
sfence
rcr dh,1
lfence
sub r10, rax
mfence
rol rbx, cl
nop
xor edi,[rbp-0x57]
```

A randomly generated sequence of instructions, and the same sequence but with randomized alignment, serialization and speculation fences added.

https://lock.cmpxchg8b.com/zenbleed.html

# Summary

- Hardware bugs
  - Deviate from specification (errata)
  - Incorrect and vague specifications

- Potential approaches to find hardware bugs
  - Manual analysis, testing
  - Fuzzing

*Program testing can be quite effective for showing the presence of bugs, but is hopelessly inadequate for showing their absence.*

–Edsger Dijkstra

THE UNIVERSITY
*of* NORTH CAROLINA
*at* CHAPEL HILL